Software Testing Tutorial

Testing Principles

a. Defect Clustering

Observation: After several rounds of testing, the QA team observes the following defect distribution:

1. **User Registration**: 10 defects
2. **Product Catalog**: 5 defects
3. **Shopping Cart**: 30 defects
4. **Payment Gateway**: 25 defects
5. **Order Management**: 8 defects

Analysis:

- The **Shopping Cart** and **Payment Gateway** modules together account for 55 defects out of a total of 78 defects found across the system.
- These two modules, therefore, contain approximately 70% of the total defects, even though they represent a smaller portion of the total modules in the system.

Software Testing

Software Testing

Defects in software have caused loss of money, time, reputation and **life**

Example:

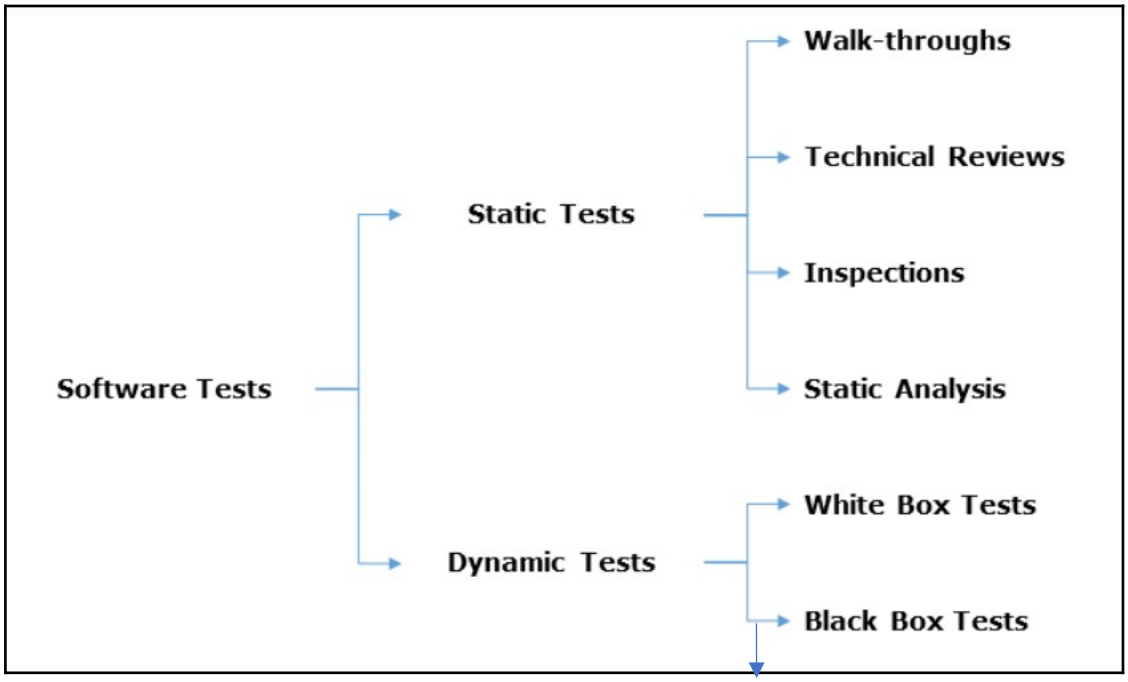
Olympic hammer throw scoring system
Loss: Loss of reputation for the International Olympic Committee.

Vancouver Stock Exchange crash
Loss: Serious financial loss for the brokers and investors, along with reputation loss for VSE

Early Warning Missile Defense System
Loss: Potential large scale loss of life and property

Two types of Tests

1. **Static Testing**
2. **Dynamic Testing**



Experienced Based Testing

b. Walkthrough vs Technical Review vs Inspection vs Static Analysis

Walkthrough	Technical Review	Inspection	Static Analysis
Informal review process	Semi formal review process	Formal review process	Can be Informal or formal review process based on who is performing it.
Used when an artefact prepared by a team belonging to one discipline needs to be reviewed by a team from another discipline.	Used when an artefact prepared by a novice in one discipline needs to be reviewed by an expert from the same discipline.	Used when a version of an artefact needs to be finalized based on a review and sign-off by a competent authority to satisfy the entry criteria for next stage in the process.	Used to analyze and predict the dynamic behavior (when executed) of an artefact which is very close to the end product (code or design) without executing it.
Requires a meeting. The creator/author of the artefact leads the activity by explaining their artefact in detail. Other participants from various teams act as audience.	Requires a meeting. The technical expert leads the activity. The creator/author aids with explanations wherever required.	Requires a meeting. A moderator leads the activity. Other participants include the artefact's creator/author and representatives from teams whose work is scheduled downstream and would depend on the correctness and completeness of the artefact in question.	Does not require a meeting always. Can be performed independently by the creator/author, his/her peers or an expert to predict failure points before the artefact can be executed.
Defects/review comments are communicated orally to the author/creator.	Defects/review comments might be communicated orally or documented for future reference and re-review.	Defects/review comments are documented, tracked periodically until they are brought to closure.	Defects/review comments might be documented if the review is performed by someone other than the creator/author.

c. Black Box Testing vs White Box Testing

Black Box Testing	White Box Testing
The software is validated by the appearance and behavior and outputs of its User Interface (UI). There is no visibility into the actual underlying code.	The software is validated by the functionality of its program modules by reviewing the code or monitoring the state of the software and its resources at different points in the program flow during execution. There is complete access and visibility into the code.
Validations are done from the software users' perspective (for compliance with the requirements.)	Validations are done from the programmers' perspective (for compliance with the programming standards, best practices and design.)
The objective is to identify the presence of defects in the underlying code. (not locating it)	The objectives can be • To predict UI behavior • To locate a defect • To validate a fixed defect • To check for compliance with design and programming standards.

d. Quality Assurance vs Quality Control

Quality Assurance	Quality Control
QA is the implementation of processes, methodologies and standards that ensure that the software developed will be up to the required quality standards.	QC is the set of activities that are carried out to verify the developed product meets the required standards.
QA focuses on the improvement of process and methodologies used to develop product.	QC focuses on the improvement of the product by identifying the bugs and issues.
It is process oriented.	It is product oriented.
QA aim is to prevent the defect.	QC aim is to identify and improve the defects.
QA is the technique of managing quality.	QC is a method to verify quality.
QA does not involve executing the program.	QC always involves executing the program

Quality Assurance is a verification activity that verifies you are doing the right thing in the right manner.	Quality assurances is a validation activity that validates the product against the requirements.
All team members are responsible for QA.	Testing team is responsible for QC
Examples of quality assurance activities include process checklists, process standards, process documentation and project audit.	Examples of quality control activities include inspection, deliverable peer reviews and the software testing process.

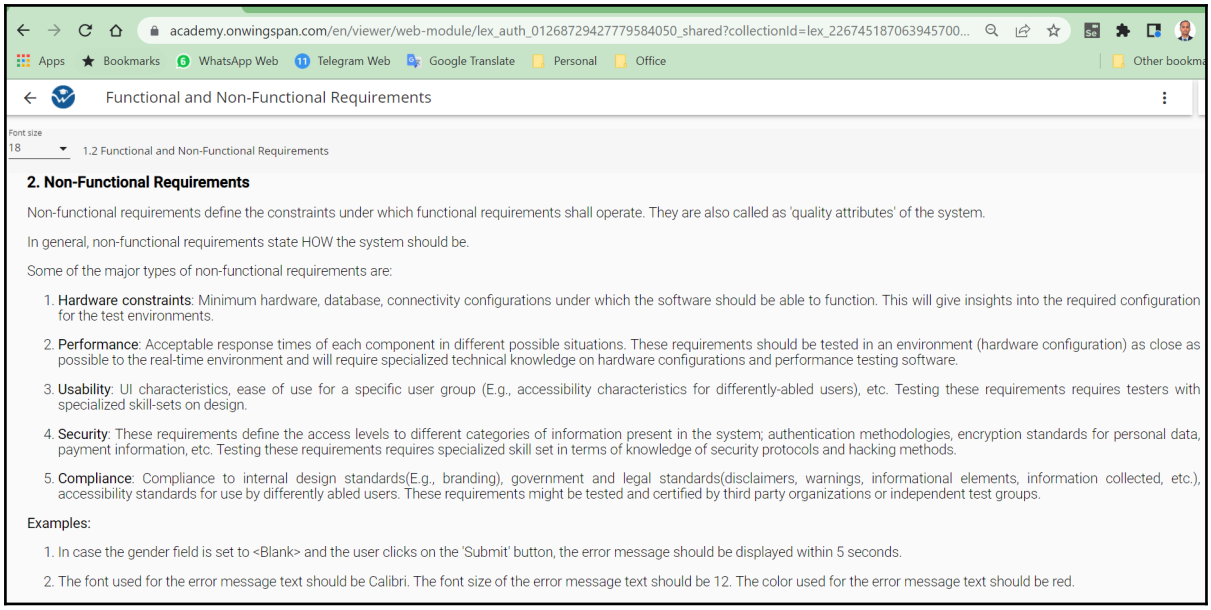
e. Alpha Testing vs Beta Testing

Types of UAT :

	Alpha Testing	Beta Testing
Performed By	Actual users who are also part of the project development team or organization.	Actual users who are not part of the project development team or organization.
Test Environment	Project development site in a controlled environment.	Actual user/customer site in real-time environment.

Requirement Analysis Phase

f. Non-Functional Requirements



g. Case Study to identify requirements as good or bad

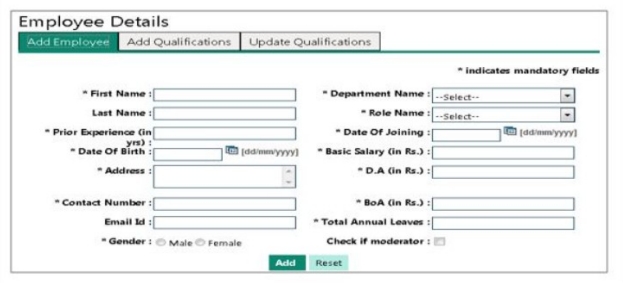
Problem Statement:

Objective

To understand requirement characteristics by reviewing and classifying the requirement statements as good or bad.

Problem Statement

Read through some of the requirements specified for the 'Add Employee' module of Health Management System that needs to be built as per the figure given below and classify them as good or bad depending on the various characteristics of software requirements.



Application level Requirements:

1. All the field names marked with an '*' before them in the look up diagram are supposed to be considered as mandatory fields.

2. A pop-up calendar function should be implemented for all date fields, using which the user can click and select the required date value to be entered in the corresponding date field.

3. The page and its fields should be user friendly.

4. The employee details report must be auto generated by the system and sent to the HMS Admin.

Field Level Requirements:

1. There must be a field "First Name" of type textbox. It should not allow the user to enter more than 30 characters.

2. There should be a field "Date of Birth" of type textbox. It should accept only a date value as input and only in the date format *mm/dd/yyyy*. If an invalid date value or an invalid format is entered, then an error message "Invalid Date of Birth" should be displayed.

3. There should be a 'Reset' field of type button. Upon clicking this button

1. All existing values in all the text fields in the page should be cleared and fields should be set to blank.

2. All existing selections in the drop-down fields, radio button fields and check box fields should be discarded and the fields have to go back to their unselected state.

3. All existing values in the date fields must be cleared and the fields should be set to blanks

4. The system must take the first two digits of the 'Contact Number' as STD code.

Solution

Once you have finished the classification, you can validate your answers here.

Solution to Exercise 2 – Requirement Characteristics

Application Level Requirement # 1

This is a bad requirement because it's incomplete. There is nothing specifically stated about

1. How the 'mandatory' ness has to be implemented. There are multiple ways in which this functionality can be implemented like error messages, warning messages, pop up boxes, disabling the 'Add' button or a combination of options described.
2. There is no mention about which fields should be mandatory.

Application Level Requirement # 2

This is a good requirement.

Application Level Requirement # 3

This is a bad requirement as 'User Friendly' is a very ambiguous term and can mean different things to users, programmers and testers. There is no baseline using which you, as a tester, can validate this requirement.

Application Level Requirement # 4

This is a bad requirement since it is incomplete. It lacks the information like

1. In what frequency the report should be generated
2. The file type of the report
3. The exact content of the report in terms of data from which fields.
4. Using which channel of communication the report should be shared with the HMS Admin.

Field Level Requirement # 1

This is a bad requirement since it is ambiguous. The ambiguity exists on how the system is supposed to work.

The possibilities are:

1. The system shall not let the user enter more than 30 characters
2. The system shall truncate the entered string to 30 characters
3. The system shall display an error message if the user enters more than 30 characters

Field Level Requirement # 1

This is a bad requirement since it is ambiguous. The ambiguity exists on how the system is supposed to work.

The possibilities are:

1. The system shall not let the user enter more than 30 characters
2. The system shall truncate the entered string to 30 characters
3. The system shall display an error message if the user enters more than 30 characters

Field Level Requirement # 2

Though this requirement looks complete, unambiguous and testable, it is in conflict with page level requirement # 2 for the date fields. It will confuse the developer and tester as to whether only a calendar can be used to input date values or they have to be typed as text or both should be allowed. Hence this is an inconsistent requirement.

Field Level Requirement # 3

This is a good requirement.

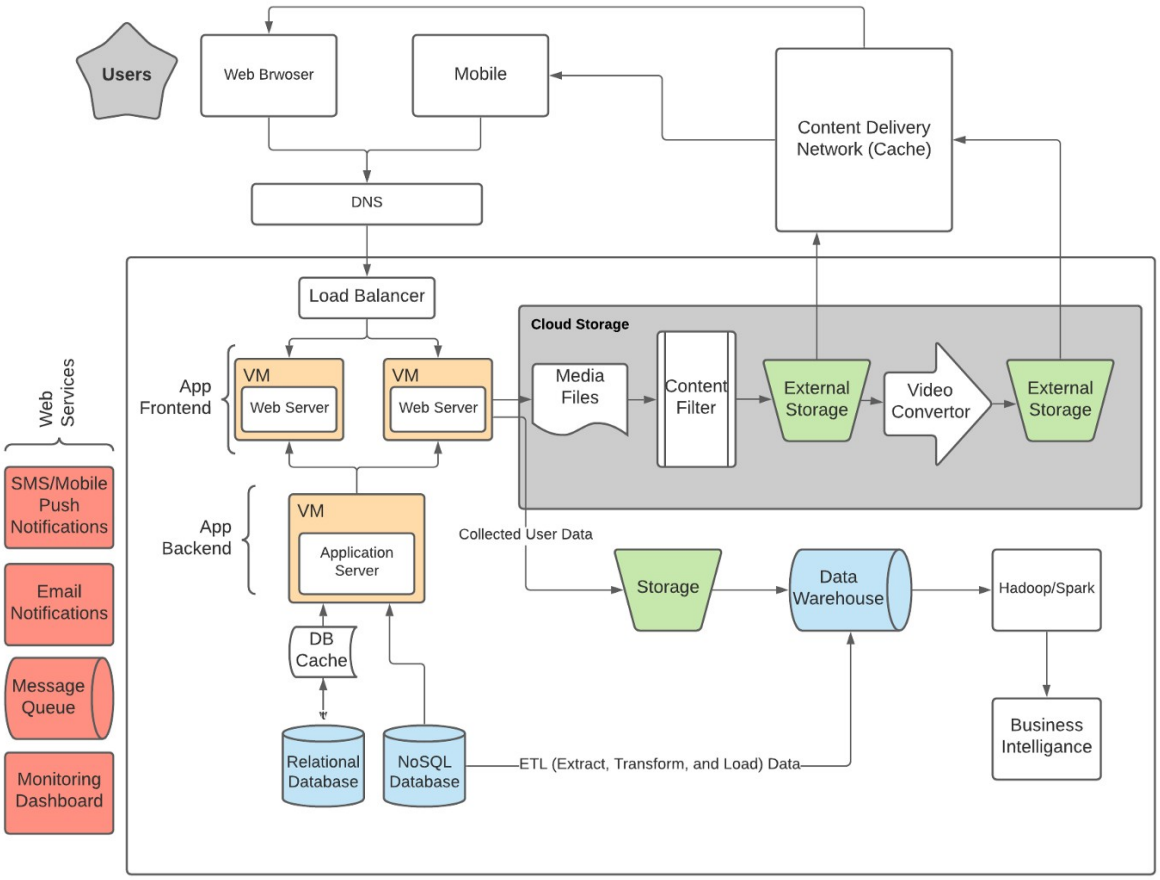
Field Level Requirement # 4

This is an incomplete requirement as there is no data provided on the type of the field, the size of the field or valid and invalid values for the field.

Sample risk in testing

Design Phase

h. Sample High Level Design

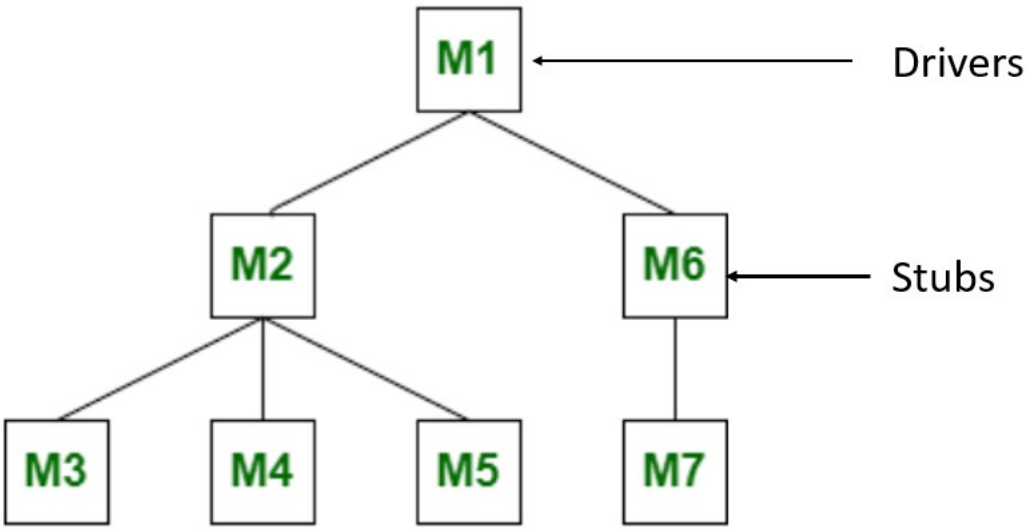


Testing Levels

1. Unit Testing
2. Integration Testing
    a. Unit Integration Testing
    b. System Integration Testing
3. System Testing
4. User Acceptance Testing

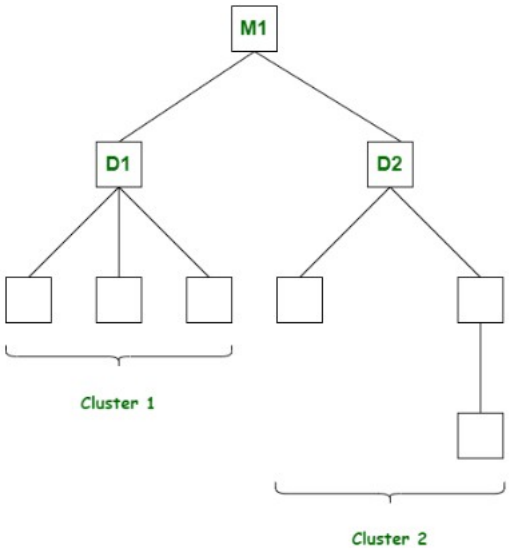
i. Top Down

In the top-down approach, the stubs are used to simulate the submodule, which implies that the Stub works as a momentary replacement. On the other hand, in the bottom-up testing approach, the drivers simulate the main module, which means that the Driver works as a momentary replacement.



Program Structure

j. Bottom-up Approach



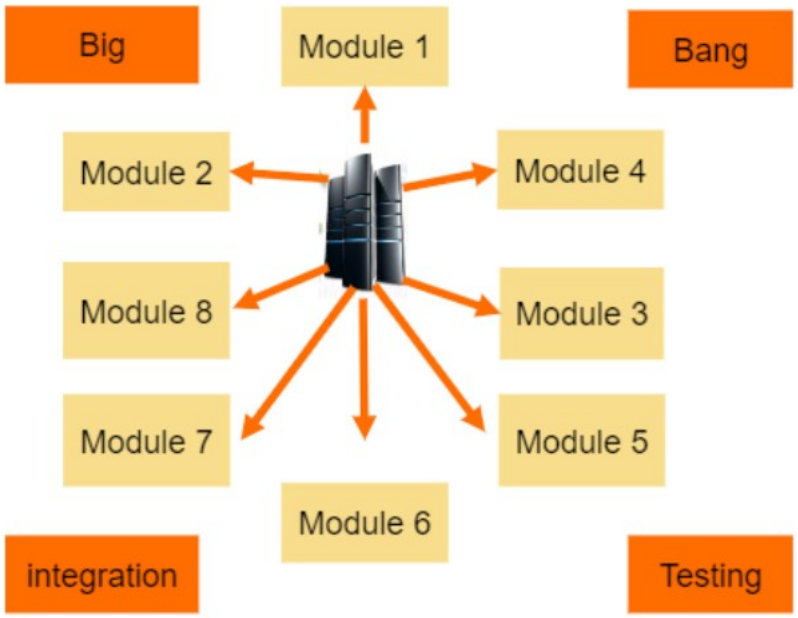
Bottom Up Integration Testing

k. Stub:

A stub is a small program routine that **substitutes for a longer program, possibly to be loaded later or that is located remotely**. For example, a program that uses Remote Procedure Calls (RPC) is compiled with stubs that substitute for the program that provides a requested procedure.
An example can be **an object that needs to grab some data from the database to respond to a method call**

l. Big Bang Approach

Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.



m. Sandwitch Apprach

Sandwich testing is a hybrid of the bottom-up and top-down approaches. Meaning, both top-level modules and lower-level modules are integrated and tested together as a system. Therefore, sandwich testing is also known as hybrid integration testing

System Integration Testing

- Carried out to check if the system works in conjunction with the other systems/interfaces(as below) and the data flows across them as desired:
  - LAN/WAN, communications middleware
  - Other internal systems (billing, stock, personnel, overnight batch, branch offices, other countries)
  - External systems (stock exchange, news, suppliers)
  - Intranet, internet / www
  - Third party packages
  - Electronic data interchange (EDI)
  - Printers
- Generally performed after System Testing or in parallel

Test Plan

n. Items to be included in Test Plan Document

Contents of a Test Plan Document

As per IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, a Test Plan Document needs to possess the following information.

Test Plan Identifier: A number, name or an alphanumeric code that uniquely identifies the Test Plan Document among all other project related documents.

Introduction: A high-level outline of the AUT and necessity for the present testing activity.

Test Items: The software and/or its specific components which have to be tested.

Features to be tested: Parts/Sections of the Software Requirements Specifications that are to be tested.

Features not to be tested: Parts/Sections of the Software Requirements Specifications that are NOT to be tested and reasons for it.

Approach: Types of tests that are to be done for specific software components or specific requirements and the intention for choosing that specific approach.

Item Pass/Fail Criteria: Pass/fail criteria defined at the level of software components or requirement sections or test types.

Item Pass/Fail Criteria: Pass/fail criteria defined at the level of software components or requirement sections or test types.

Suspension criteria and resumption criteria: A condition (or a combination of conditions) that would affect the testing tasks to an extent that they would be suspended. A condition or combination of conditions that are required to be met to resume suspended tests.

Test deliverables: Documents, artifacts and proofs that need to be delivered at specific milestones during the course of the testing phase.

Test tasks: Diverse kinds of testing tasks that must be performed, the execution order and their respective entry and exit criteria.

Environmental needs: The hardware and software configurations that the testing team needs along with their required dates and timelines to achieve the testing objectives as planned.

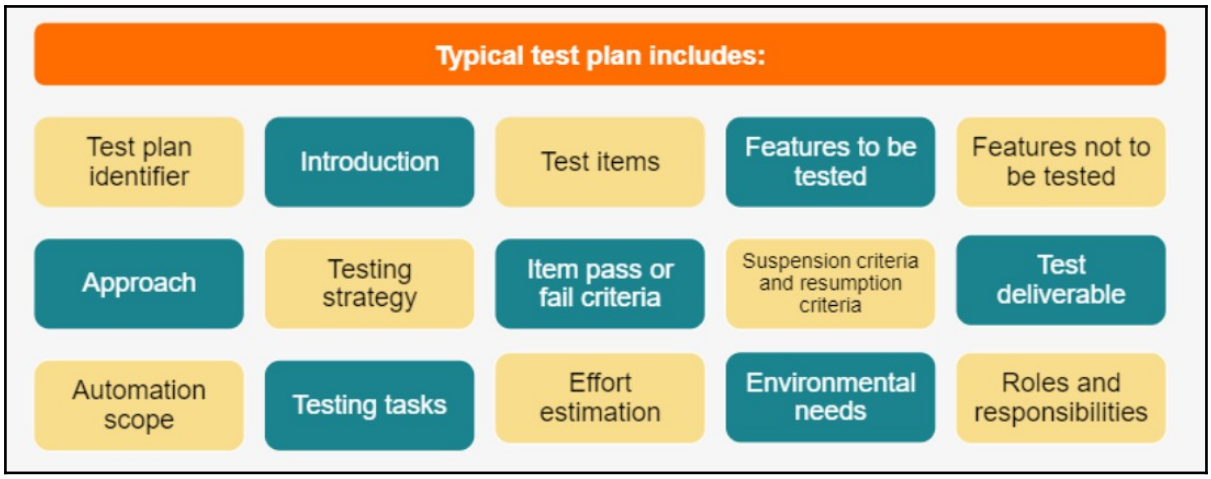
Staffing and training needs: The number of testers needed to perform all the testing tasks based on the estimated testing effort, the skills that they require, their current skill levels and training plans to cover the skill gaps, if any.

Responsibilities: Specific responsibilities of different project team members and stake holders.

Schedule: The planned start and end dates of different testing tasks in the STLC with defined milestones at different points in time.

Planning risks and contingencies: List of possible risks during the STLC, their probability of occurrence, their impact and the contingency plan to reduce or eliminate their impact, if they occur.

Approvals: This section is used to record the approvals given for the test plan by different stakeholders of the project along with the dates and versions that were approved.

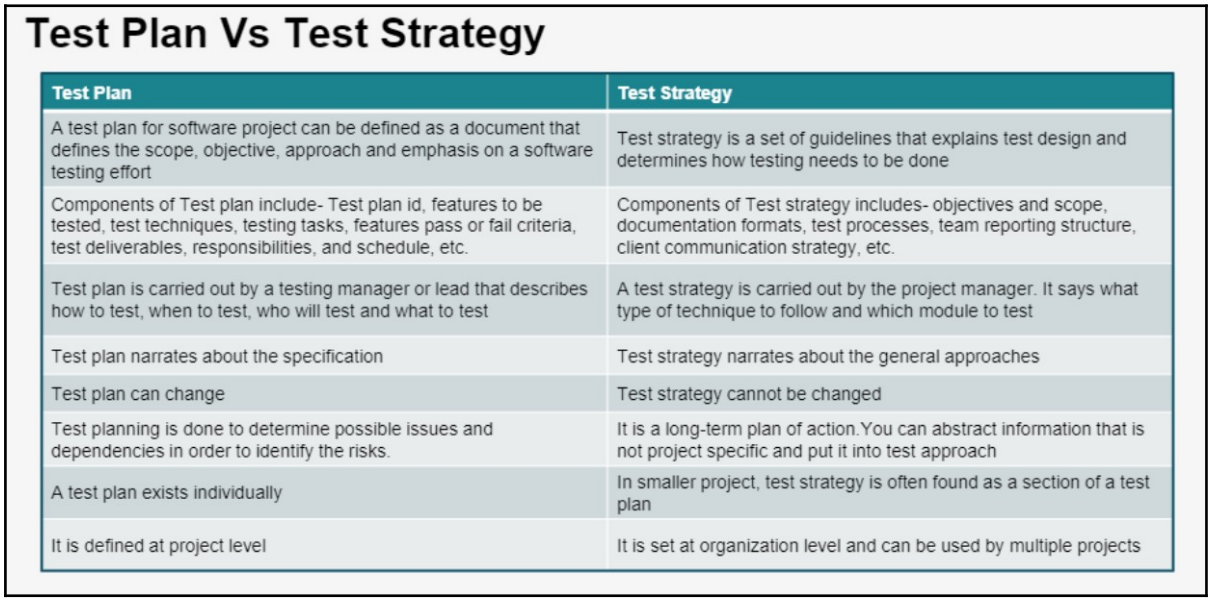


Test Plan vs Test Strategy

Parameter	Test Plan	Test Strategy
Definition	A test plan is a document that encompasses the scope and different activities involved in the testing process.	A test strategy is a high-level document that encompasses guidelines and principles involved in carrying out the testing process.
Goal	The primary goal here is to define how to test a product, what to test it on when to test it, who will test, and who will verify the results.	Here, the primary goal is to define the principles to be followed during the testing process.
Purpose	It is done to identify possible inconsistencies in the final	It is a plan of action of the testing process on a long-term basis.

Parameter	Test Plan	Test Strategy
	product and mitigate them through the testing process.	
Level	It is used on a project level only.	It is used on an organizational level.
Repetition	It is used by one project only and is very rarely repeated.	It is used by multiple projects and can be repeated a lot of times.
Change Susceptibility	It is a dynamic document and can be changed repeatedly depending on testing specifications.	It can not be changed since it is a static document.
Components	Its components include o **Objective** o **Scope** o **Features to be tested** o **Features not to be tested** o **Test methodology** o **Approach** o **Roles and responsibility** o **Schedule** o **Bug tracking** o **Entry and exit criteria** o **Test automation** o **Effort estimation,** o **Test deliverables** o **Assumption** o **Risk** o **Mitigation plan** o **Templets**	Its components include – o **Scope and overview** o **Testing tools** o **Testing metrics** o **Requirement Traceability Matrix** o **Training plan,** o **Business issues** o **Team Reporting Structure** o **Change and configuration management** o **Test summary**
Performed by	Usually, a testing manager or lead creates a test plan that outlines how to test, when to test, who will test, and what will be tested.	A test strategy is developed by the project manager as part of the testing process. The document specifies what type of technique to use and which modules should be tested.
Scope	It thoroughly defines the whole testing activities.	It only focuses on high-level test methods and strategies.
Derivation	It is derived with the help of Use case documents, software requirement specification documents, and product description documents.	It is derived from the business requirement specification document.

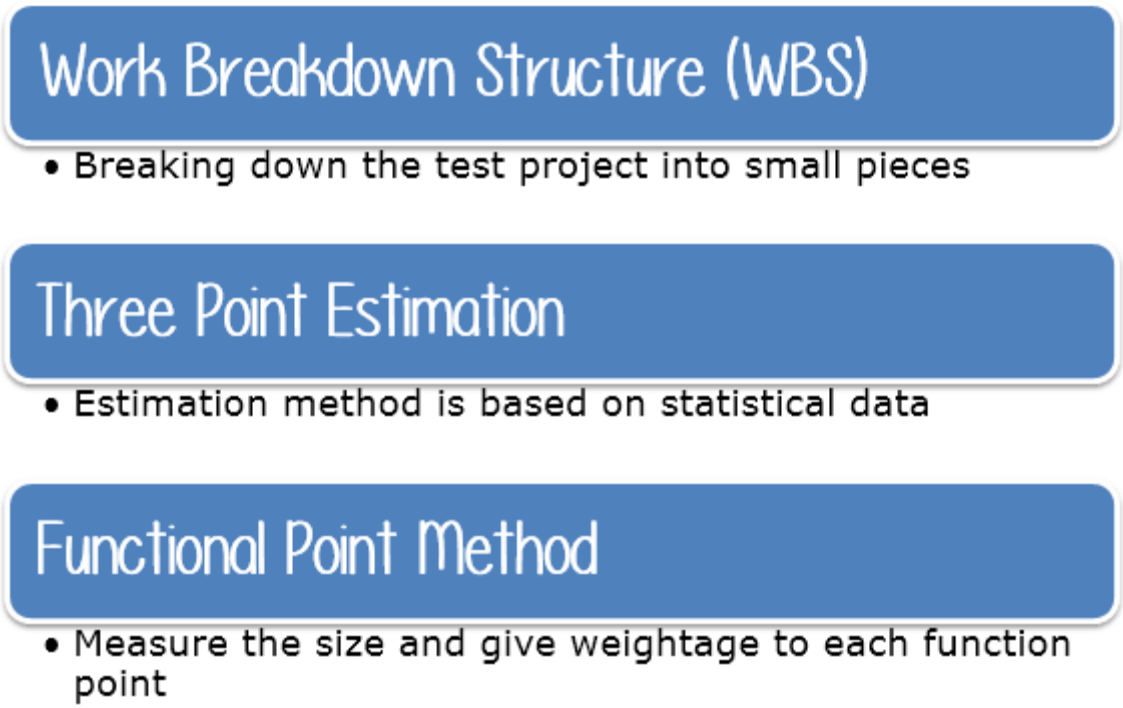
Parameter	Test Plan	Test Strategy
Existence	A test plan is usually found to exist individually.	A test strategy can be a part of test plans, in some cases like smaller projects.
Basis	It is based on testing strategies.	It is based on standards that have been pre-defined.
Types	Level-specific test plans, Type-specific test plans, and Master test plans are the different types of test plans.	Analytical strategy, Model-based strategy, Methodical strategy, Standard-compliant strategy, Reactive strategy, Consultative strategy, and Regression averse strategy are the different types of test strategies.
Influence	It affects only a single project at a time.	It affects many projects at a time.
Narration	It narrates the common specifications in the testing of a particular project.	It narrates the approaches in testing.



Testing Estimation Technique

- Work Breakdown Structure
- 3-Point Software Testing Estimation Technique

- Wide band Delphi technique
- Function Point/Testing Point Analysis
- Use Case Point Method
- Percentage distribution
- Ad-hoc method



In three-point estimation, **three** values are produced initially for every task based on **prior experience** or **best-guesses** as follows

Test Case Design

o. Effective and Efficient Testing

If you achieve 100% test coverage, it gives the confidence that you have encountered all possible defects that are existing in the software, at least once.

Coverage Effectiveness

- A test suite design is said to **effective** if 100% test coverage is achieved by its test cases.
- Exhaustive testing (testing a software using all probable data inputs and configurations) can ensure 100% test coverage but it is impossible as it would take months (or even years).

Coverage Efficiency

- If you take exit criteria (when the testing should stop) from the test plan document into account, then you will understand that you have to achieve 100% test coverage with as less number test cases as possible.
- A test suite design is said to be **efficient** if it uses the least number of tests possible to achieve 100% test coverage.

Test case optimization is the process limiting the number of test cases, on the basis of sound principles and scientific assumptions, required to achieve maximum test coverage.

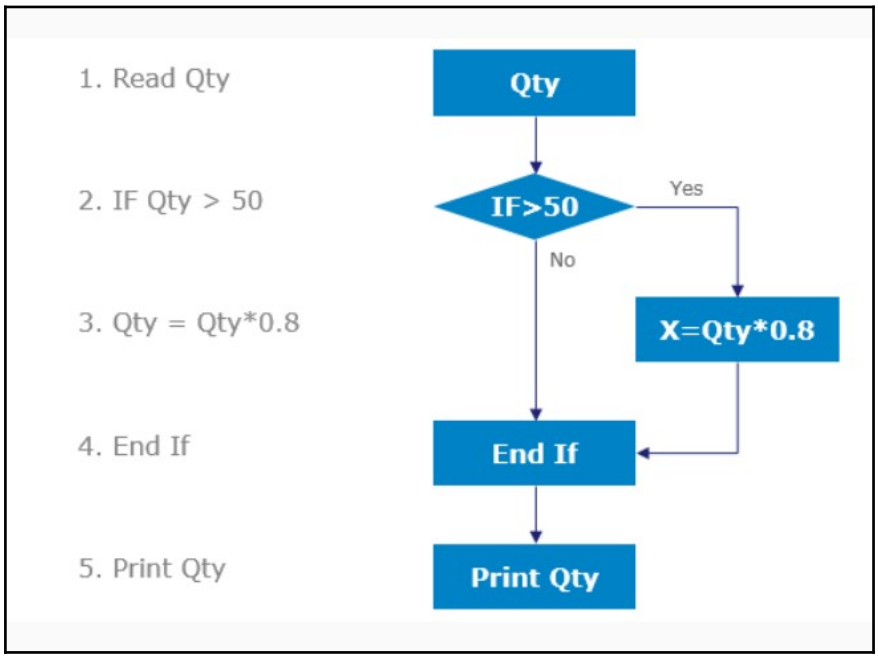
The techniques that testers use to optimize their test cases can be classified into three groups, as shown below:

- Structure based techniques
- Specification based techniques
- Experience techniques

White Box Testing / Structure Based Testing:

p. Statement Coverage

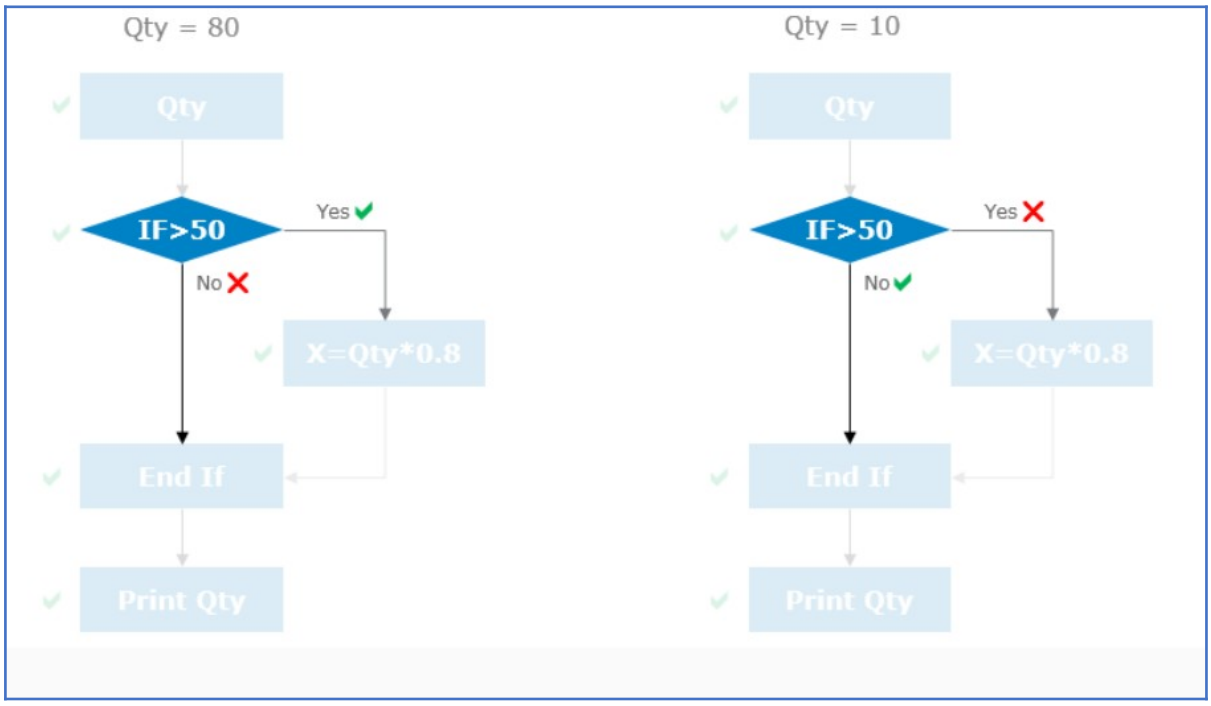
Find the minimum no of test cases which executes all the statement at least once

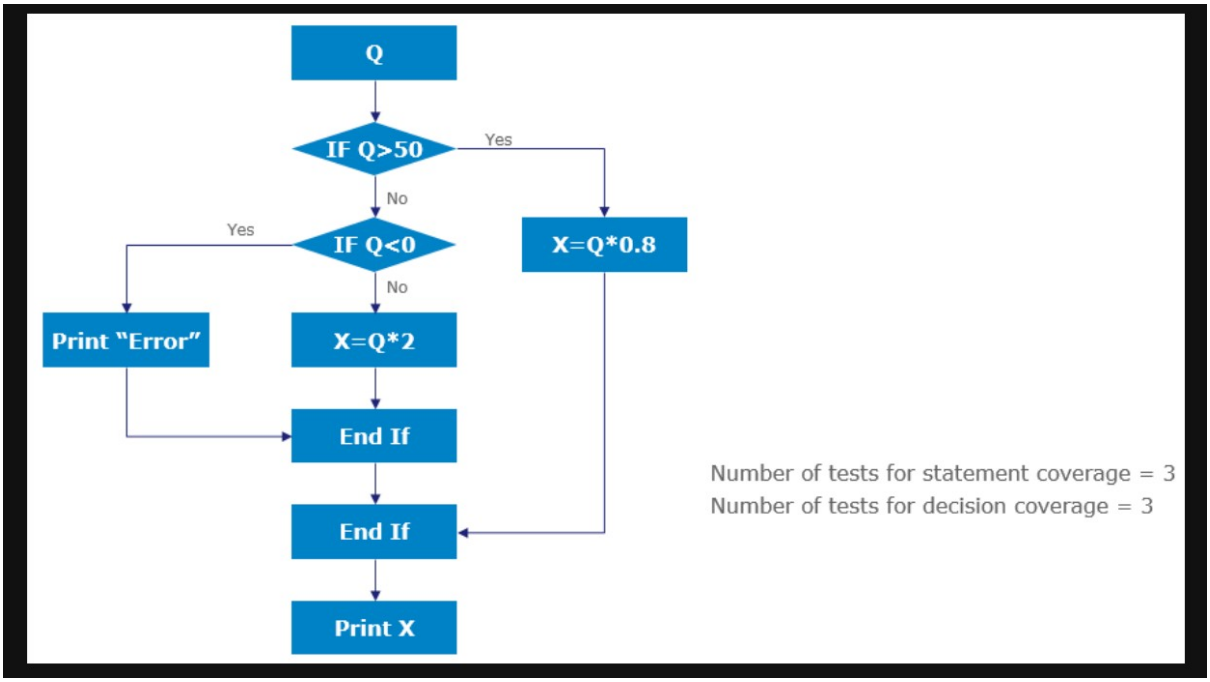


q. Decision Coverage

Find the minimum no of test cases which covers all possible decisions

Decision Coverage can also be called branch testing or control flow testing





Case Study 2:

Problem Statement:

```
1. Read NumberOfExams
2. While NumberOfExams != 0
3.       Read Mark
4.       TotalMarks = TotalMarks + Mark
5. End While
6. Print TotalMarks
```

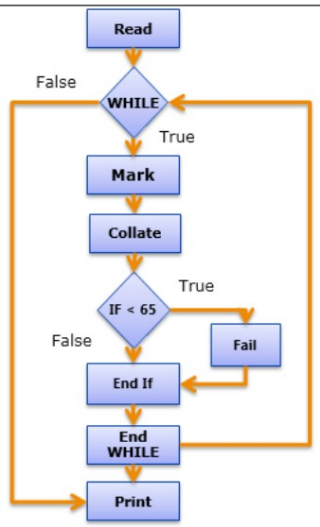
For the above pseudo code, find the number of tests necessary to attain

1. 100% statement coverage
2. 100% decision coverage

Case Study 3:

Problem Statement:

1. Read Number of Exams
2. WHILE more exams DO
3. Mark answers
4. Collate total marks
5. IF Score < 65
6. Fail = Fail + 1
7. End If
8. END WHILE
9. Print "No more to mark"



Solution to Exercise 3 – Statement Coverage and Decision Coverage in a While Loop with 'If' condition inside

The number of tests required to achieve

1. 100% statement coverage - Answer: 1
2. 100% decision coverage - Answer: 1

Note: For while loops, you can achieve 100% statement coverage and 100% decision coverage using only 1 value.

r. Condition Coverage:

Complex software like a life insurance underwriting system can either accept or reject a policy enrollment request from a customer based on combination of a number of inputs - Income, age, already existing policies, number of claims in previous policies, existing health conditions etc. The customer is not only evaluated based on each individual data point collected, but also conditional combinations of such inputs.

A decision statement of a such computer programs can involve complex boolean expressions which have

- Multiple input values
- Multiple relational evaluations between the inputs

Such expressions are called decision predicates.

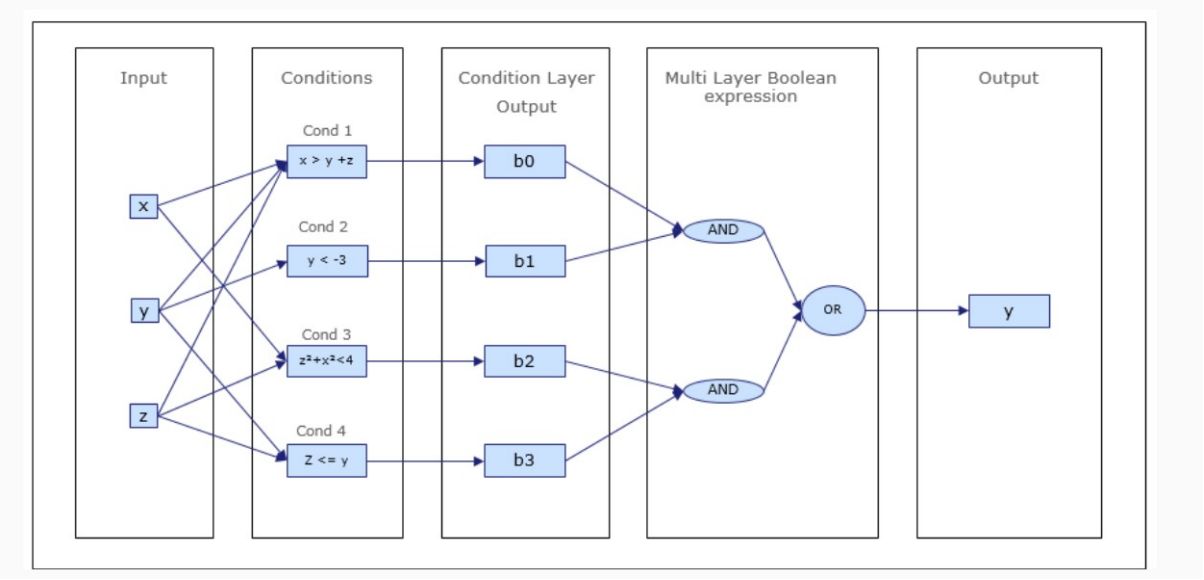
Here's an example of a decision predicate.

```
1. b_out = ((x>y+z) AND (y<-3)) OR ((z²+x²<4) AND (z≤y))
```

Even though there are multiple inputs and multiple evaluations, the final result can branch the program flow only in two ways (b_out = true, b_out = false). Here, testing all possible decisions (2) is not enough. We need to test all possible ways in which the inputs arrive at either of the decisions.

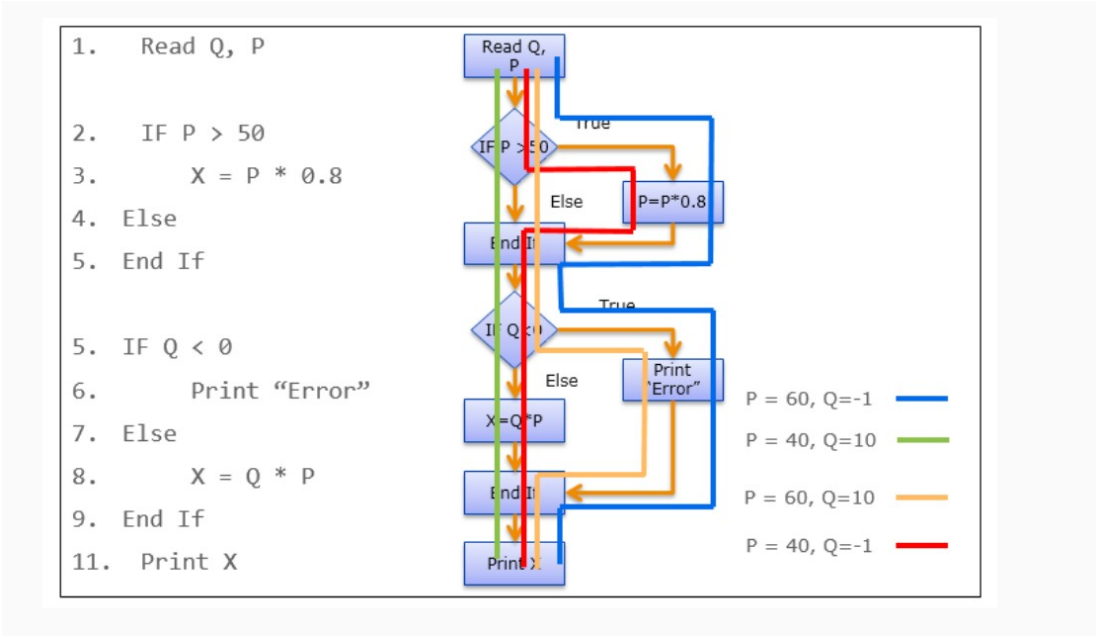
For scenarios like these, we use condition coverage techniques. In this technique, the goal is to cover all possible conditional evaluations of the boolean expressions.

Lets see how we can derive tests out of the decision predicate given in the example above, to achieve 100% decision coverage.



s. Path Coverage

Paths refer to the number of ways in which the program execution can flow through the program code.



Black Box Testing / Specification Based Testing

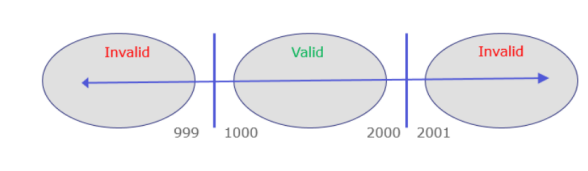
Given below are the major types of specification based test design techniques:

- Equivalence partitioning
- Boundary value analysis (BVA)
- Decision Table
- State transition testing
- Orthogonal Array
- Use case testing

t. Equivalence Partitioning

Example 1

To test a software component whose requirement specification says that acceptable input values must be >=1000 AND <=2000, the equivalence classes of input values would be



- Less than 1000 (Negative test. Expected outcome is that the input should be rejected)
- Between 1000 and 2000 (Positive test. Expected outcome is that the input should be accepted)
- Greater than 2000 (Negative test. Expected outcome is that the input should be rejected)

Thus this requirement can be tested using only 3 test cases by picking 1 input value from each equivalence class for every test case.

Example 2

To test a software component whose requirement specification says that the field must accept only letters of the alphabet, the equivalence classes of input values would be

- Alphabetic characters
- Numbers
- Special Characters

Again, this requirement can be tested using only 3 test cases.

u. Boundary Value Analysis

v. Decision Table

w. State Transition Diagram

x. Orthogonal Array Testing

y. Use Case Testing

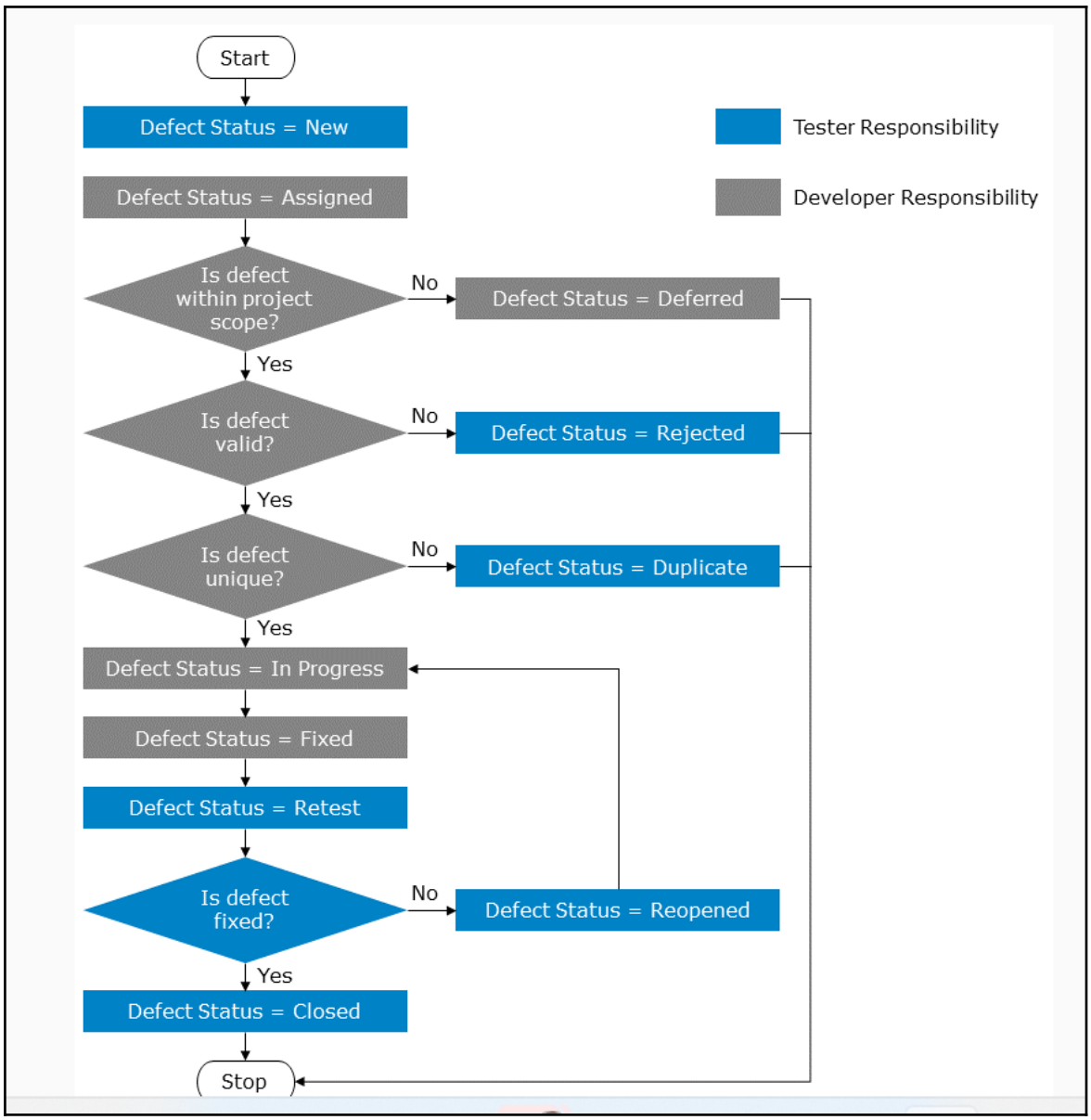
z. Exploratory Testing

aa. Experience Based Testing

Test Execution Phase

bb. Defect

Defect Status:



cc. Assigning Severity and Priority to the Defects

To understand the difference between the concepts of priority and severity of defects

Problem statement

Assign the priority and severity values for the following defects. The values of priority can be assigned as HIGH, MEDIUM and LOW. The values of severity can be assigned as CRITICAL, MAJOR and MINOR

1. On providing the URL of the application in the address bar of the browser and hitting ENTER key, the application is not invoked. The error message "The page cannot be displayed" is coming up.
2. On entering all the customer details in the web page and clicking on 'Submit' button, the page is not getting refreshed and no action seems to be happening. This anomaly is happening only when using Mozilla Firefox however the functionality is working fine in Internet Explorer and Google Chrome.
3. The date of birth field in the customer information page is accepting values of future dates also. As per the requirements, its input values should have been restricted only to past dates.
4. In the home page of the web application, the image of the company logo is not loaded in the top right corner as per the requirements. Instead that space is empty.
5. In the application, the text fields sizes are different according to their maximum length allowed. These fields, though in a single column, do not start and end at the same margins. This is causing a misalignment of the fields and they are not appearing in a neat and organized way.

7 Principles of Software Testing

Background

It is important that you achieve optimum test results while conducting software testing without deviating from the goal. But how you determine that you are following the right strategy for testing? For that, you need to stick to some basic testing principles. Here are the common seven testing principles that are widely practiced in the software industry.

1) Testing shows presence of defects

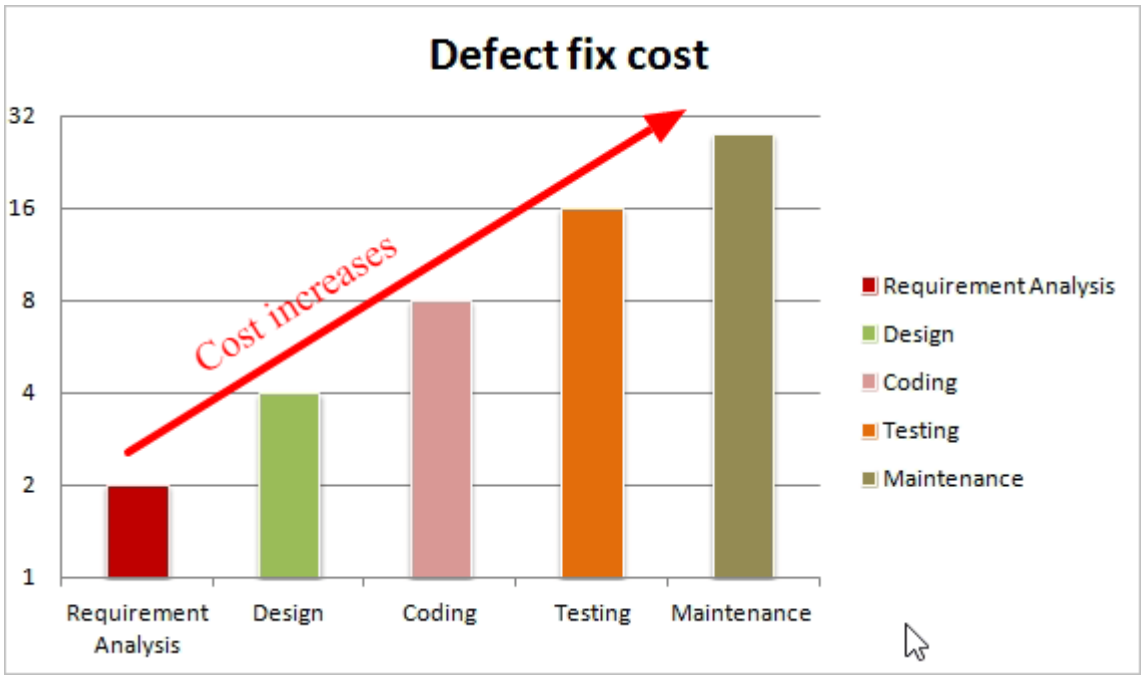
By testing you can show presence of defects in a product but you can never prove that the product under test is defect free. The testing strategy should focus on clustering the defects in order to reduce the residual risk of the software to a minimum but, even if no defects are found, it is not a proof of correctness.

2) Exhaustive testing is impossible

Testing all possible scenarios (all combinations of preconditions and inputs) is not feasible. Techniques like risk analysis should be used to prioritize and focus the testing efforts.

3) Early testing

The testing activities should be started as early as possible during the software or system development life cycle. The cost to fix a bug increases exponentially if the bug is found in a later phase of the development life cycle. The testing activity shall be focused on defined objectives.

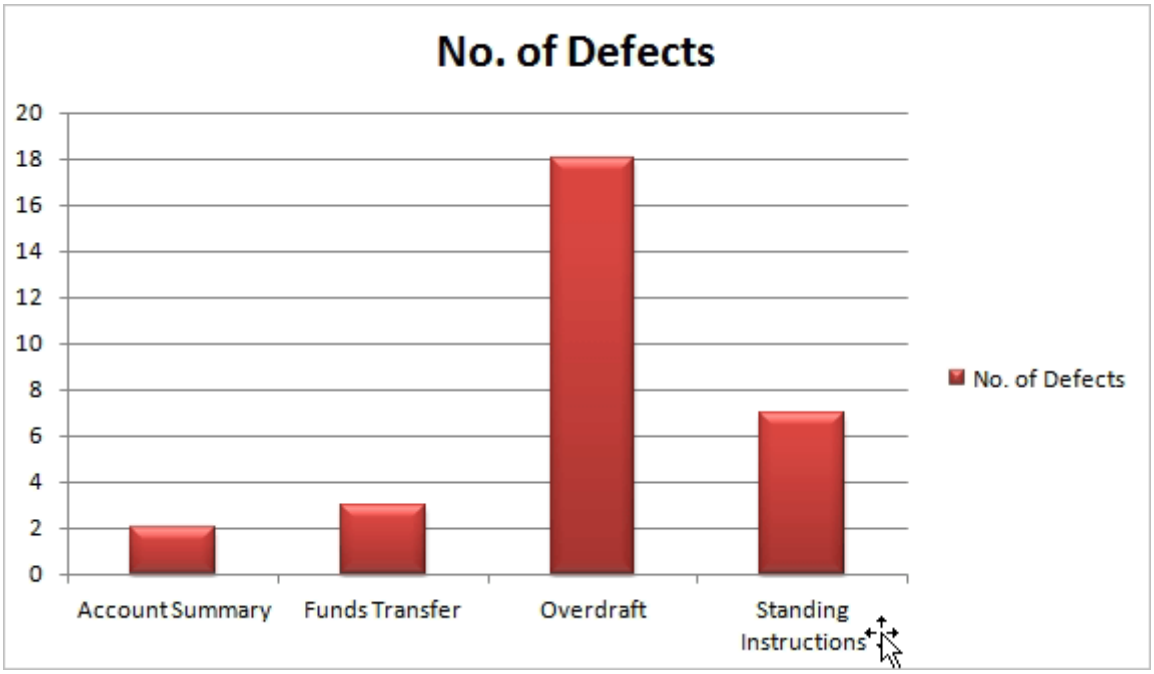


4) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.



5) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide. Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let's see this video of the public launch of Windows 98

You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

6) Testing is Context-Dependent

There are several domains available in the market like Banking, Insurance, Medical, Travel, Advertisement etc and each domain has a number of applications. Also for each domain, their applications have different requirements, functions, different testing purpose, risk, techniques etc.

Different domains are tested differently, thus testing is purely based on the context of the domain or application.

For Example, testing a banking application is different than testing any e-commerce or advertising application. The risk associated with each type of application is different, thus it is not effective to use the same method, technique, and testing type to test all types of application.

7) Absence of Error – fallacy
It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.
In short words: if a systems does not fulfill the user needs and expectations then having a bug free system does not help.

To solve this problem, the another principle of testing states that Early Testing.

Statement Coverage:

```
Int i =10;

If (i>0){
i = i+1
Print (i);
Continue;
j=i+2;
Print(j)
}
```

What is Decision Coverage or Branch Coverage?

Decision coverage or Branch coverage is **a testing method**, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed. That is, every decision is taken each way, true and false.

```
Int i =-10;
Int j =-20;
```

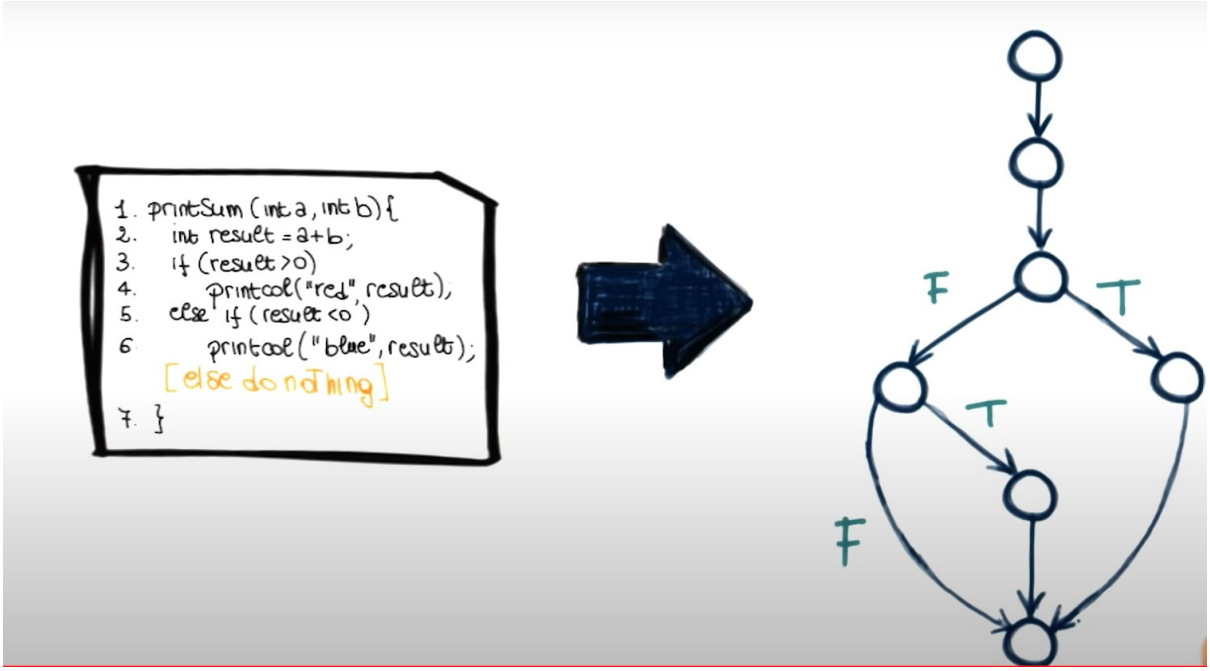
```
If ( i>0  || j > 10 ){  // if ( i>0 && j>10)
i = i+1
Print (i);
j=i+2;
Print(j)
}else {

}
```

1. True, False
2. False , True
3. ~~True, True~~
4. False, False

Condition Coverage
The percentage of conditions within decision expressions that have been evaluated to both true and false. Note that 100% condition coverage does not guarantee 100% decision coverage.
For example, “if (A || B) {do something} else {do something else}” is tested with [0 1], [1 0], then A and B will both have been evaluated to 0 and 1, but the else branch will not be taken because neither test leaves both A and B false.

Path Coverage:
Measures whether all possible ways through a given piece of code have been implemented and tested. One should test all the outputs to test all the paths.

Control flow testing is a type of software testing that uses program’s control flow as a model. Control flow testing is a structural testing strategy. This testing technique comes under white box testing. For the type of control flow testing, all the structure, design, code and implementation of the software should be known to the testing team.



Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects. Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

Error vs Defect vs Bug vs Failure

A mistake in coding is called Error, error found by tester is called Defect, defect accepted by development team then it is called Bug, build does not meet the requirements then it Is Failure

Different Types of Functional Testing

1. Smoke Testing

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not. Smoke testing is a confirmation for QA team to proceed with further software testing. It consists of a minimal set of tests run on each build to test software functionalities. Smoke testing is also known as “Build Verification Testing” or “Confidence Testing.”

In computer programming and software testing, **smoke testing** (also **confidence testing**, **sanity testing**,[1] **build verification test** (**BVT**)[2][3][4] and **build acceptance test**) is preliminary testing to reveal simple failures severe enough to, for example, reject a prospective software release. Smoke tests are a subset of test cases that cover the most important functionality of a component or system, used to aid assessment of whether main functions of the software appear to work correctly.[1][2] When used to determine if a computer program should be subjected to further, more fine-grained testing, a smoke test may be called an **intake test.**

Smoke Testing comes into the picture at the time of receiving build software from the development team. The purpose of smoke testing is to determine whether the build software is testable or not. It is done at the time of "building software." This process is also known as "Day 0".

dd. 2.Sanity Testing

Sanity testing, a software testing technique performed by the test team for some basic tests. The aim of basic test is to be conducted whenever a new build is received for testing. The terminologies such as Smoke Test or Build Verification Test or Basic Acceptance Test or Sanity Test are interchangeably used, however, each one of them is used under a slightly different scenario.

Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change.

Sanity testing can be narrow and deep. Sanity test is a narrow regression test that focuses on one or a few areas of functionality.

Sanity Testing is a subset of regression testing. Sanity testing is performed to ensure that the code changes that are made are working as properly. Sanity testing is a stoppage to check whether

testing for the build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on build where the production deployment is required immediately like a critical bug fix.

Definition: Sanity testing is a subset of regression testing. After receiving the software build, sanity testing is performed to ensure that the code changes introduced are working as expected. This testing is a checkpoint to determine if testing for the build can proceed or not.

3. Integration Testing

Integration testing is a type of testing where individual software modules are logically integrated and tested as a group. This testing is done to test that individual modules work as expected when they are combined.

Each individual module (units) is first tested in isolation. Once unit testing has been performed, these units are integrated. The integration testing is then performed to validate that the modules work as expected when combined.

Integration testing is not necessarily performed at the end of software development. Instead, it is performed throughout the development life cycle as the individual modules get developed.

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

Integration testing (sometimes called **integration and testing**, abbreviated **I&T**) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.[1] It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.[2]

4.Big Bang Integration

Big Bang Integration Testing is an approach in which all software components (modules) are combined at once and make a complicated system. This unity of different modules is then tested as an entity. According to this checking method, the integration process will not be executed until all components are completed.

Big Bang Integration Testing is an integration testing strategy, wherein all units are linked at once, which results in a complete and efficient system. In this type of integration testing all the components as well as the modules of the software are integrated simultaneously, after which everything is tested as a whole. During the process of big bang integration testing, most of the developed modules are coupled together to form a complete software system or a major part of the system, which is then used for integration testing. This approach of software testing is very effective as it enables software testers to save time as well as their efforts during the integration testing process.

Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

5.Top Down Approach

A top-down approach is used when management wants to improve overall reliability and/or does not know what the principal causes of problems may be. If a facility manager notes that production losses through unanticipated downtime are increasing, or maintenance costs for the whole facility have increased, he will probably call for a top-down analysis. He may also authorize this type of analysis when he suspects that there is a pervasive root cause problem, such as inadequate operator training, that is affecting many facility subsystems.

In simple terms, a top-down approach is an investment strategy that selects various sectors or industries and tries to achieve a balance in an investment portfolio. The top-down approach analyzes the risk by aggregating the impact of internal operational failures. It measures the variances in the economic variables that are not explained by the external macro-economic factors. As such, this approach is simple and

not data-intensive. The top-down approach relies mainly on historical data. This approach is opposite to the bottom-up approach.

Definition: a Top-down Approach is an autocratic and hierarchical style of decision making, organizational change and leadership, in which strategies or plans are first conceived by one or a few top managers, and then disseminated (cascaded) further down the organizational structure. The lower levels in the hierarchy are, to a greater or lesser extent, bound by the decisions of the top management.

6.Bottom Down Approach

A bottom-up approach is **the piecing together of systems to give rise to more complex systems**, thus making the original systems sub-systems of the emergent system. Bottom-up processing is a type of information processing based on incoming data from the environment to form a perception.

Definition: a Bottom-up Approach is a democratic and consultative style of decision making, organizational change and leadership, in which employee participation is promoted at all levels the organization.

The bottom up approach definition is when the investing involves picking out certain securities based on how the security is priced. Bottom up approach also involves looking at the potential return and risk associated.

7. stubs and drives

A stub is like a placeholder for a software component that hasn't been fully developed yet. It mimics the behavior of the real component by providing simple, predefined responses to function calls. This allows other

parts of the system to be tested even if the actual component isn't ready or is undergoing changes.

On the other hand, a driver is like a control mechanism. It's a small program or module that interacts with the component being tested. The driver calls functions or methods in the component and supplies test data to simulate how the component would be used in real-world scenarios.

In **software testing life cycle**, there are numerous components that play a prominent part in making the process of testing accurate and hassle free. Every element related to testing strives to improve its quality and helps deliver accurate and expected results and services that are in compliance with the defined specifications. **Stubs and drivers** are two such elements used in software testing process, which act as a temporary replacement for a module. These are an integral part of software testing process as well as general software development. Therefore, to help you understand the significance of **stubs and drivers** in software testing, here is elaborated discussion on the same.

Stubs and drivers are **used to test modules**. Stubs are used to test the functionality of modules, whereas drivers are used when the main module is not ready.12-Feb-2020
Stubs and drivers both are **dummy modules** and are only created for test purposes. Stubs are used in top down testing approach, when one has the major module ready to test, but the sub modules are still not ready yet. ... Drivers are used in bottom up testing approach.

8. System Testing

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Definition: System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a pre-requisite and is done by the testing team.

System Testing means testing the system as a whole. All the modules/components are integrated in order to verify if the system works as expected or not.
System Testing is done after Integration Testing. This plays an important role in delivering a high-quality product.

9. Re-Testing

Re-testing is executing a previously failed test against new software to check if the problem is resolved. After a defect has been fixed, re-testing is performed to check the scenario under the same environmental conditions.

During Re-testing, testers look for granular details at the changed area of functionality, whereas regression testing covers all the main functions to ensure that no functionalities are broken due to this change.

Retesting is the testing of failed test cases after the bugs corresponding to those test cases have been fixed by the developers. It is planned testing and is also known as 'Confirmation Testing'.

Retesting : To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build.

Retesting is running the previously failed test cases again on the new software to verify whether the defects posted earlier are fixed or not.

In simple words, Retesting is testing a specific bug after it was fixed.

10. Regression Testing

REGRESSION TESTING is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Regression testing (rarely *non-regression testing*[1]) is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.[2] If not, that would be called a regression. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.[3] As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of tests (*non-regression analysis*[4]).

Regression Testing is a type of testing that is done to verify that a code change in the software does not impact the existing functionality of the product.

This is to ensure that the product works fine with new functionality, bug fixes or any changes to the existing feature. Previously executed test cases are re-executed in order to verify the impact of the change.

11. Accessibility Testing

Software testing involves various techniques and methods to check the effectiveness, accuracy, efficiency, result of the system.

We perform **accessibility testing** in a software application to determine whether the application being tested is useful for people with disabilities.

Accessibility testing is the practice of making your web and mobile apps usable to as many people as possible. It makes apps accessible to those with disabilities, such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

Accessibility testing works best when incorporated into your testing strategies — don't let it be an afterthought. Align it with your test cycle and sync your results all in one place with Perfecto's test reporting.

Accessibility testing is another type of software testing used to test the application from the physically challenged person's point of view. Here the physical disability could be old age, hearing, color blindness, and other

underprivileged groups. It is also known as **508 compliance** testing. In this, we will test a web application to ensure that every user can access the website.

12.User Acceptance Testing

User Acceptance Testing (UAT) is the final stage of any software development life cycle. This is when actual users test the software to see if it is able to carry out the required tasks it was designed to address in real-world situations. UAT tests adherence to customers' requirements. UAT testers aim to validate changes that were made against original requirements.

User acceptance testing, a testing methodology where the clients/end users involved in testing the product to validate the product against their requirements. It is performed at client location at developer's site.

For industry such as medicine or aviation industry, contract and regulatory compliance testing and operational acceptance testing is also carried out as part of user acceptance testing.

UAT is context dependent and the UAT plans are prepared based on the requirements and NOT mandatory to execute all kinds of user acceptance tests and even coordinated and contributed by testing team.

13.Compability Testing

A compatibility test is an assessment used to ensure a software application is properly working across different browsers, databases, operating systems (OS), mobile devices, networks and hardware. Compatibility testing is a form of non-functional software testing -- meaning it tests aspects such as usability, reliability and performance -- that is used to ensure trustworthy applications and customer satisfaction.

Compatibility tests are crucial to the successful performance of applications. They should be performed whenever a build becomes stable enough to undergo testing.

Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.

> Checking the functionality of an application on different software, hardware platforms, network, and browsers is known as compatibility testing.

14.Exploratory Testing

Exploratory testing is an approach to software testing that is often described as simultaneous learning, test design, and execution. It focuses on discovery and relies on the guidance of the individual tester to uncover defects that are not easily covered in the scope of other tests. The practice of exploratory testing has gathered momentum in recent years. Testers and QA managers are encouraged to include exploratory testing as part of a comprehensive test coverage strategy.

Exploratory Testing is a type of software testing where Test cases are not created in advance but testers check system on the fly. They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a "thinking" activity.
Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester.

"Exploratory testing" – as the name suggests, is a simultaneous learning, test design, and test execution process. We can say that in this testing test planning, analysis, design and test execution, are all done together and instantly.

This testing is about exploring the system and encouraging real-time and practical thinking of a tester.

15.Ad hoc/Random/Monkey Testing

Ad Hoc Testing is an informal and random style of testing performed by testers who are well aware of the functioning of software. It is also referred to as **Random Testing** or **Monkey Testing**. Tester may refer existing test cases and pick some

randomly to test the application. The testing steps and the scenarios depend on the tester, and defects are found by random checking.

Comparison is inevitable when it comes to exploring different testing types. It is vital to understand and employ the right combination for a complete multi-dimensional testing. In this article, we will compare Ad Hoc Testing and Exploratory testing to understand them better.

Ad hoc Testing is an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possible stage. Ad hoc testing is done randomly and it is usually an unplanned activity which does not follow any documentation and test design techniques to create test cases.

16.Alpha Testing

Definition: Alpha testing is a type of testing that is done on an application towards the end of a development process when the product is almost in a usable state.

Alpha Testing is a type of software testing performed to identify bugs before releasing the software product to the real users or public. It is a type of acceptance testing. The main objective of alpha testing is to refine the software product by finding and fixing the bugs that were not discovered through previous tests. **Alpha Testing** is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them.

17.Beta Testing

Beta testing is a type of user acceptance testing where the product team gives a nearly finished product to a group of target users to evaluate product performance in the real world.

There is no standard for what a beta test should look like and how to set up beta testing. The actual testing procedure should be relevant to your testing goals. However, there are a few requirements that a product needs to comply with in order to be ready for beta testing:

Beta testing also known as user testing takes place at the end users site by the end users to validate the usability, functionality, compatibility, and reliability testing.

Beta testing adds value to the software development life cycle as it allows the "real" customer an opportunity to provide inputs into the design, functionality, and usability of a product. These inputs are not only critical to the success of the product but also an investment into future products when the gathered data is managed effectively.

18.End to End Testing

End to end testing (E2E testing) refers to a software testing method that involves testing an application's workflow from beginning to end. This method basically aims to replicate real user scenarios so that the system can be validated for integration and data integrity.
Essentially, the test goes through every operation the application can perform to test how the application communicates with hardware, network connectivity, external dependencies, databases, and other applications. Usually, E2E testing is executed after functional and system testing is complete.

End To End Testing is a software testing method that validates entire software from starting to the end along with its integration with external interfaces. The purpose of end-to-end testing is testing whole software for dependencies, data integrity and communication with other systems, interfaces and databases to exercise complete production like scenario.

Along with the software system, it also validates batch/data processing from other upstream/downstream systems. Hence, the

name **“End-to-End”**. End to End Testing is usually executed after functional and System Testing. It uses actual production like data and test environment to simulate real-time settings. End-to-End testing is also called **Chain Testing**.

19.Risk Based Testing

The most common complaint that comes from software testing using the Agile method is the lack of time. Since the term is itself a synonym for speed, its emphasis on getting things done is self-explanatory. However, this brings up a burning question for every Agile tester in the world.

Risk-based testing (RBT) is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in software, based on the risk of failure, the function of their importance and likelihood or impact of failure.[1][2][3][4] In theory, there are an infinite number of possible tests. Risk-based testing uses risk (re-)assessments to steer all phases of the test process, i.e., test planning, test design, test implementation, test execution and test evaluation.[5] This includes for instance, ranking of tests, and subtests, for functionality; test techniques such as boundary-value analysis, all-pairs testing and state transition tables aim to find the areas most likely to be defective.

Risk-based testing (RBT) is essentially a test performed for projects depending on the risks. Risk-based testing strategies make use of risks to prioritize and highlight the right tests at the time of test execution.

Considering that there might not be ample time to check all kinds of functionality, risk-based testing mainly concentrates on testing the functionality that carries the biggest impact and the possibility of failure.

20.Parallel Testing

Parallel testing is a semi-automated testing process that relies on cloud technology and virtualization to perform tests against several configurations at the same time. The goal of this process is to resolve the limitations of time and budget while still assuring quality.

Parallel Testing is a software testing type in which multiple versions or subcomponents of an application are tested with same

input on different systems simultaneously to reduce test execution time. The purpose of parallel testing is finding out if legacy version and new version are behaving the same or differently and ensuring whether new version is more efficient or not.

Parallel testing is testing multiple applications or components of the application concurrently, to reduce the test time. Parallel tests consist of two or more parts that check separate parts or features of an application. These parts are executed on individual computers simultaneously. Parallel testing allows reducing the test time significantly and increase the testing efficiency.

21.Concurrent Testing

Concurrency testing is also known as multi-user testing, performed to identify the defects in an application when multiple users login to the application.

It helps in identifying and measuring the problems in system parameters such as response time, throughput, locks/dead locks or any other issues associated with concurrency.

Concurrency testing is mainly used to check the performance of a website when there are multiple users active on your website. That's why it is also called as Multi-User Testing. Synchronization testing is like a step to get a website's traffic ready, So that it doesn't get stuck when there are multiple users. In other words, we can say monitoring the effect while multiple users take the same action at the same time.

Concurrent testing is performed to test the stability of a software product when the multiple number of users access and utilise its services, at a same time. This helps in getting a rough idea of the problems arising out of multiple usages such as increased response time, crashes, deadlocks and throughputs. Also known by the name of **multi-user testing**, the basic purpose of carrying out the concurrent testing, over a software product, is to evaluate and assess its concurrency quality, so as to deal with the load of multiple users, accessing and using the software product, simultaneously, in an efficient and effective manner.

Types of Severity

*In Software Testing, **Types of Severity** of bug/defect can be categorized into four parts :*

- **Critical / Showstopper**: This defect indicates complete shut-down of the process, nothing can proceed further
- **Major/ High**: It is a highly severe defect and collapses the system. However, certain parts of the system remain functional
- **Moderate / Medium**: It causes some undesirable behaviour, but the system is still functional
- **Low / Cosmetic**: It won't cause any major break-down of the system

Priority Types

***Types of Priority** of bug/defect can be categorized into three parts :*

- **High:** The defect must be resolved as soon as possible as it affects the system severely and cannot be used until it is fixed
- **Medium:** During the normal course of the development activities defect should be resolved. It can wait until a new version is created
- **Low:** The Defect is an irritant but repair can be done once the more serious Defect has been fixed

Differences Between Bug Leakage and Bug Release

Aspect	Bug Leakage	Bug Release
Definition	Bugs that go undetected and reach production.	Acceptance of known bugs in a released software.
Focus	Detection failure during testing.	Decision-making regarding the release of software.
Impact	Leads to customer dissatisfaction and potential revenue loss.	Can be communicated to users with potential workarounds.
Timing	Occurs when software is already in production.	Happens before the software is officially deployed.

Smoke Testing vs Sanity Testing

Re-testing vs Regression Testing

TDD vs BDD vs ATDD

Parameters	TDD	BDD	ATDD
Definition	TDD streamlines the software delivery process by developing the test case before the developers begin writing the code	BDD provides full test coverage based on the expected system behavior.	ATDD uses acceptance test cases to provide predetermined functionality of the system and fulfill end users' requirements.
Key Focus	Feature implementation; unit tests	System behavior, understanding of requirements	Defining precise requirements, writing Acceptance Tests

Parties involved	Developers	Customers, Developers, QAs	Customers, Developers, QAs
Ease of understanding	Since tests are technical, it is difficult for a non-technical person to understand it	Easy for a non-technical person to understand it	Easy for a non-technical person to understand it
Languages used	Programming languages like Python, Java, etc.,	Simple plain English, Gherkin	Simple plain English, Gherkin
Tools used	Cucumber, JDave, JBehave, BeanSpec, Gherkin Concordian, Spec Flow, Junit, TestNG, FitNesse, NUnit frameworks, Selenium tool (any open source tools)	Gherkin, Dave, RSpec, Behat, Lettuce, Specflow, BeanSpec, MSpec, JBehave, Concordian, Cucumber with Serenity / Selenium	FitNesse, TestNG, Spectacular, Concordian, EasyB, Robot Framework, Thucydides, FIT
Project compatibility	Projects that don't need the involvement of end users like API, server, etc.	Projects in which the user is handling e-commerce websites or different types of apps.	Projects targeted at customer experience; Projects with high competition like apps and e-commerce websites
Bugs	Can be easily tracked down and reduced the occurrence	More difficult to track down compared to TDD	Difficult for developers to track them down